Tempel, Michael

Article

Generative art for all

Journal of Innovation and Entrepreneurship

Provided in Cooperation with:
Springer Nature

Suggested Citation: Tempel, Michael (2017) : Generative art for all, Journal of Innovation and Entrepreneurship, ISSN 2192-5372, Springer, Heidelberg, Vol. 6, Iss. 12, pp. 1-10, https://doi.org/10.1186/s13731-017-0072-1

This Version is available at:
https://hdl.handle.net/10419/194857

Tempel *Journal of Innovation and Entrepreneurship* (2017) 6:12
DOI 10.1186/s13731-017-0072-1

Journal of Innovation and Entrepreneurship

SHORT REPORT Open Access

Generative art for all

Michael Tempel

Correspondence:
michaelt@media.mit.edu;
http://www.logofoundation.org
Logo Foundation, 250 West 85th Street Suite 4D, 10024 New York, NY, USA

Abstract

Background: Generative art is created by a system that operates autonomously, or semi-autonomously, rather than directly by the artist. The artist creates the system and establishes parameters that affect the outcome, but the outcome itself emerges from the system rather than from the artist. Generative art systems are frequently computer programs, although biological, social, or other systems may also be used as well.

Findings: Computer programming environments are often technically demanding, but there are also those that are more accessible and offer novices ways to engage with concepts and practices of generative art. We report on our experience with two such environments, TurtleArt and Scratch, that we have used in workshops with preservice and in-service teachers over the past several years.

Conclusions: TurtleArt and Scratch are two programming environments that are accessible to novices and provide a way to explore and create works of generative art.

Keywords: Generative art, Algorithmic art, Computer programming, Coding, Scratch, TurtleArt, Logo

Generative art refers to art that is created by a system that operates autonomously (Galanter 2003; McCormack et al. 2014). The artist may create the system, and/or set some parameters that affect the outcome, but the result is created, at least in part, by the system rather than directly by the artist. Generative art systems are frequently computer programs, although biological, social, or other systems may also be used to generate art.

Art that is created by using a computer is not necessarily generative. If one is using a paint or drawing application to create an image, the computer is a tool—much like a pencil or paint brush—that is controlled directly by the artist. The application is not acting autonomously.

A related concept is algorithmic art, which may be considered as one type of generative art. It generally refers to art that is created via an algorithm, implemented as a computer program, determining the outcome. But artwork involving symmetry and pattern may be implicitly algorithmic regardless of how it is created.[1] The artist is following a step by step sequence of rules, even if the algorithm is not spelled out explicitly. In this sense, algorithmic art has existed for thousands of years. For example, consider how you would create a floor tile pattern from individual white, black, and brown hexagonal tiles (Fig. 1).

You might follow this algorithm:

1. Place a black tile in the middle of the floor.
2. Surround the black tile with white tiles.
3. Surround the white tiles with brown tiles.
4. Etc.

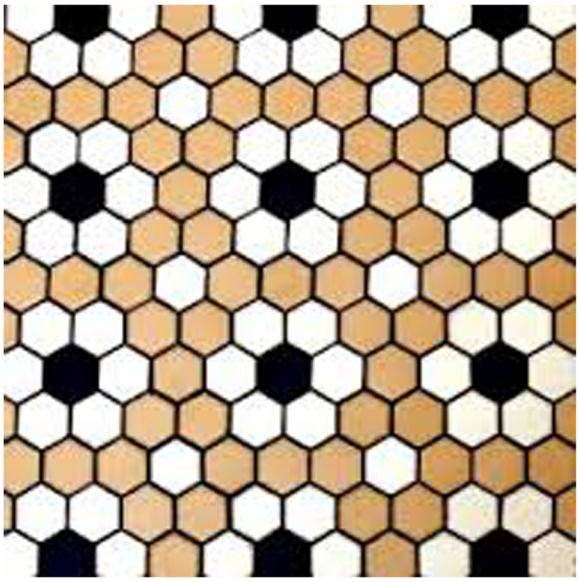

Fig. 1 Hexagonal floor tile pattern. This pattern may be created from individual tiles by following an implicit or explicit algorithm

In practice, you probably would not think the steps through explicitly, but would simply look at a picture of the desired result and proceed to lay out the tiles to duplicate the picture. You would implicitly be following an algorithm, either the one above or any one of a number of other possible algorithms that would generate the same pattern.

In reality, such a floor is generally not assembled out of individual tiles. Instead, the tiles come in sheets that are about one foot square. The tiles, with the pattern in place, are attached to a flexible mesh. These sheets are then put together to form the floor. The sheets are manufactured by a machine that is programmed to produce the pattern. In this case, the algorithm must be made explicit so as to be able to instruct the machine.

Even in algorithmic art, there is room for uncertainty and surprises. A musical score is an algorithm that specifies how a piece should be played. Yet different performances of the same piece vary in ways that are noticeable to listeners. This is in part due to performers not following the score exactly, but there are also aspects of performance that are not captured in the written score and come from the artist.

An algorithm may determine a result very precisely, but there may still be surprises. The image in Fig. 2, drawn by a program written in TurtleArt, which we will elaborate on below, includes only straight lines. There are no curves. Figure 3 shows the code that produced the image.

Generative art often has an element of uncertainty. This may be due to the inclusion of randomness in the algorithm used to produce the result, but can also be the result of the unpredictable nature of some parameter such as the number of people viewing the artwork, or the price of crude oil. Generative art systems may be very complex. For example, Electric Sheep (http://www.electricsheep.org/) is running on thousands of computers generating animations, or "sheep," that morph and reproduce based on algorithms, but also affected by the popularity of individual sheep among members the worldwide community of users.

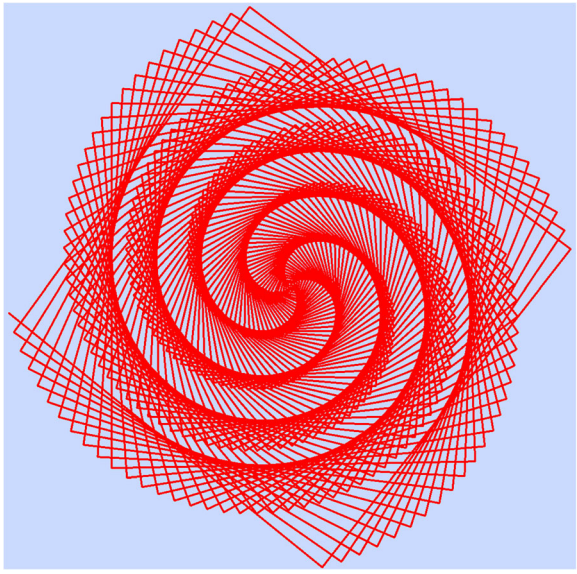

Fig. 2 Spiral drawn with TurtleArt. The image appears to have curves, but is drawn using only straight lines

A challenge for educators is to make a generative art experience available to young students and to people with limited technical expertise. Can we construct accessible creative environments that bring the learner/artist in touch with the concepts around generative art? Such an introduction should lay the groundwork for people to move to mainstream systems more comfortably if they choose to do so. We will look at two such environments, TurtleArt[2] and Scratch[3], that we have used in workshops over the past several years.[4]

Long live the turtle

TurtleArt is a contemporary version of the Logo programming language with a Logo-speaking Turtle as its starring character. Logo development began a half century ago,

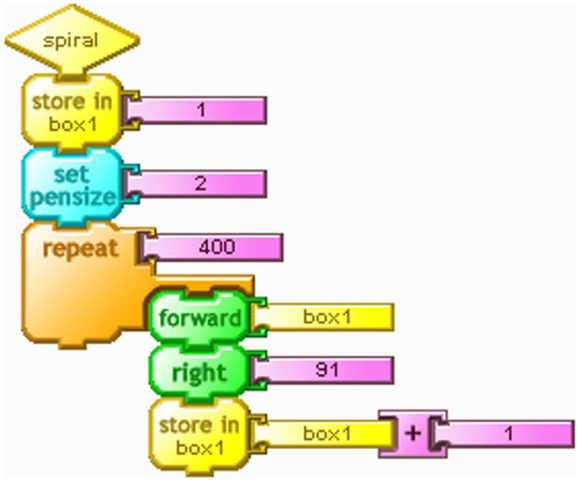

Fig. 3 TurtleArt code to draw an apparent spiral. Lines are drawn only by the *forward* block. The length of each line is determined by the value of the variable *box1*, which is incremented by one step in each iteration of the *repeat* loop. The *right* block rotates the turtle 91° clockwise, while not changing the turtle's position

and there have been more than 300 versions implemented to date (What is Logo? 2014; Boytchev 2016). Most of these include Turtle Geometry, which was conceived of as a form of geometry that was more accessible to young learners than the traditional school geometries of Euclid and Descartes (Papert, Seymour 1981).

The Turtle was originally a robotic creature that traveled around on the floor in response to commands from a computer to move forward and back, and to turn to the right or left. It had a pen, positioned vertically in its center, which when down, allowed the Turtle to draw as it moved. By the early 1970s, the turtle had migrated to the computer screen. Looking at a screen turtle is like having a view from above of a robot floor turtle (Figs. 4 and 5).

The goal for most Logo users has been to learn programming and mathematics, usually with visual results. But there has also been a long tradition of Logo programming where creating a visual effect is the goal and programming is the means.

With TurtleArt, the aim is to create visual art, specifically drawings that may be viewed on the computer screen or printed, framed, and hung on the wall. Programming and Turtle Geometry are the means by which those drawing are created.

TurtleArt is entirely algorithmic. The only way to generate an image on the screen is by writing and executing a program. This is in contrast to paint and draw applications in which images are created and manipulated directly using a mouse or touchpad. There are also environments, Scratch (https://scratch.mit.edu/) and MicroWorlds (www.microworlds.com), for example, which incorporate both approaches.

TurtleArt uses Blocks Programming (Tempel 2013) in which programs are created by snapping together graphical pieces on the screen, like putting together a jigsaw puzzle. The function of each block and the overall structure of the program are expressed

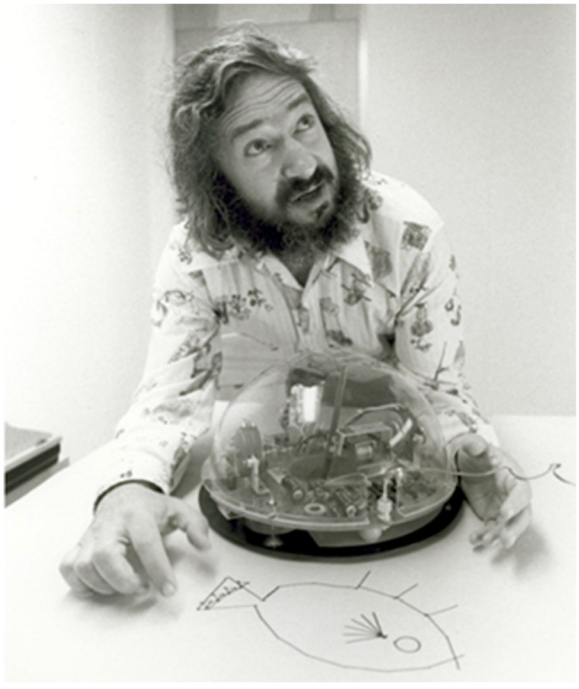

Fig. 4 Seymour Papert with a robotic turtle, c.1972

Fig. 5 An image drawn by a screen turtle

by the shapes and colors of the blocks, and how they fit together. These visual clues help make a program more understandable.

Another advantage of this style of programming, which is especially important for beginners, is that in contrast to conventional text-based languages, it is impossible to make syntax errors. Over the past few years, Blocks Programming has become widely used, driven largely by the popularity of Scratch.

The TurtleArt snippet of code in Fig. 6 draws a square of 100 pixels on a side.

The size of the turtle's pen may be specified. The square is drawn with the pen set to its default size of 4 pixels. The line in Fig. 7 is drawn with the pen set to a width of 50 pixels.

One can also set the pen's color, with 0 being red, 90 purple, and the rest of the rainbow in between. Shade may be set as well, with a number ranging from 0 to 100. Lower numbers are very dark, and higher numbers are pastel shades. The default value is 50.

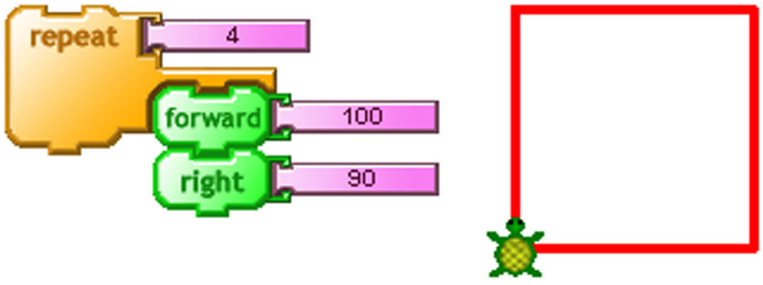

Fig. 6 TurtleArt image of a square and the code that drew it

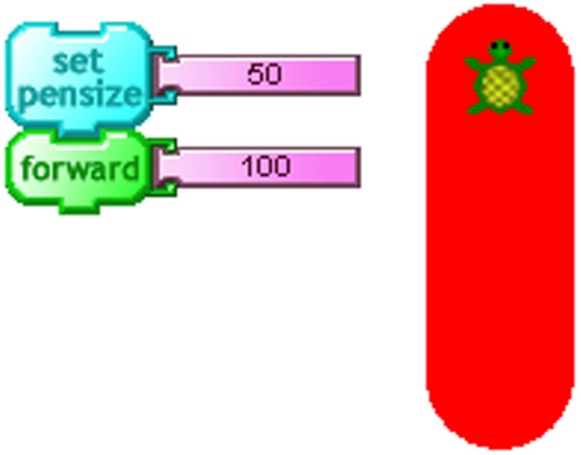

Fig. 7 The turtle's pen is 50 pixels wide

By manipulating these three parameters—pen size, color, and shade—a wide variety of visual effects can be achieved. A TurtleArt program may be entirely deterministic. The image that is created is the same every time the program is run. But there may be an element of randomness in the program so that the image is generated somewhat differently each time. The example in Fig. 8 is called Titanic Rain[5].

The droplets are all in the same size and shape, but vary in color. Figure 9 shows the code that draws a single droplet.

The pen size is set to 70, which is fairly wide. The shade is set to 30, which is a bit on the dark side. Then, the turtle takes 70 steps forward. With each step, the pen size is narrowed by one pixel and the shade is lightened. The result is that the droplet starts out fat and dark, but lightens and tapers to a point as the turtle move up the canvass.

So far, there is no randomness in the algorithm. It is introduced in the code that creates the full image (Fig. 10).

First, the screen is filled with black. Then, 60 droplets are drawn. The color of each is set randomly in a range of blue through purple to red. The position of each droplet is set to a random *XY* coordinate.[6] Running the program multiple times will produce

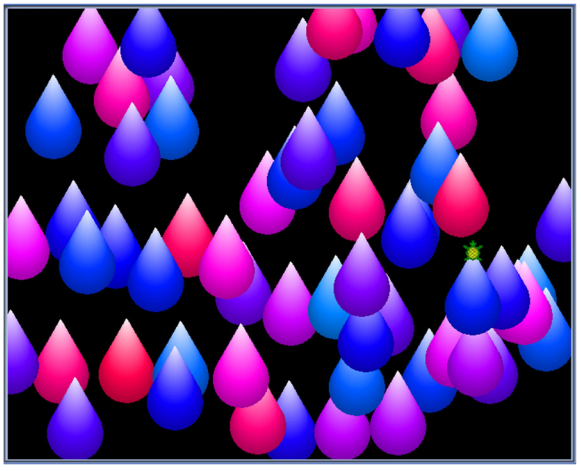

Fig. 8 Titanic Rain

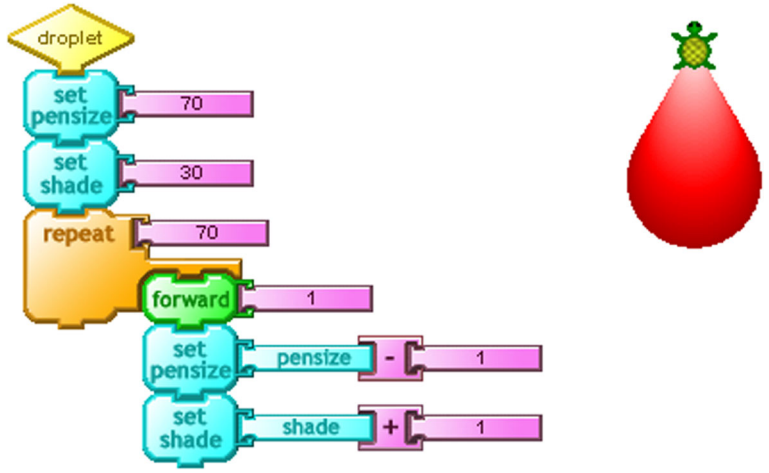

Fig. 9 The code that draws one droplet in Titanic Rain

similar, but different images. There will be variation in the placement of the droplets, and some results will tend more to the red, while others further toward blue or purple.

In this next example, called Patio (Fig. 11), we see a regular pattern of five rows of six squares.

There is no randomness in this arrangement on the XY grid. There is some randomness in the elements, with slight variations in color and shade. And some squares are lined up with the edges of the canvass, but most are rotated a bit clockwise or counterclockwise.

These examples bring us back to the earlier discussion about different styles of geometry. The turtle is egocentric. It moves and turns relative to where it is and which way it is pointing. It does not know or care about the larger context. The floor turtle can be

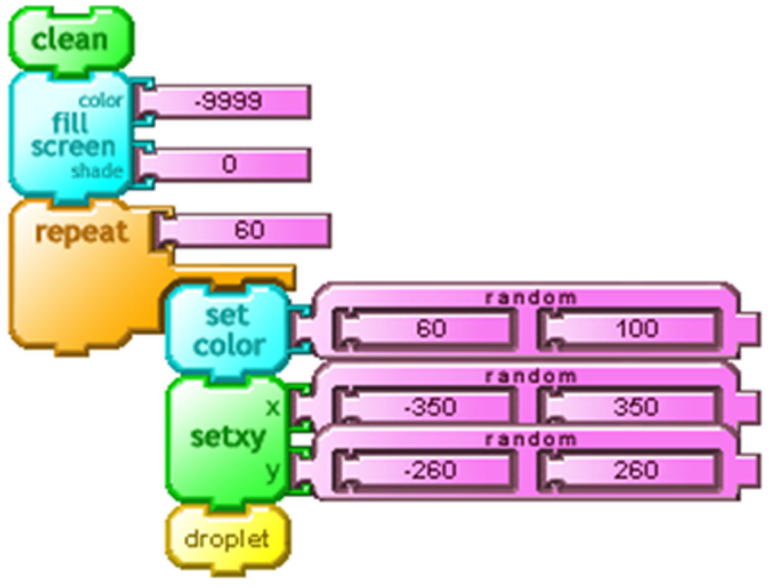

Fig. 10 Code to draw Titanic Rain

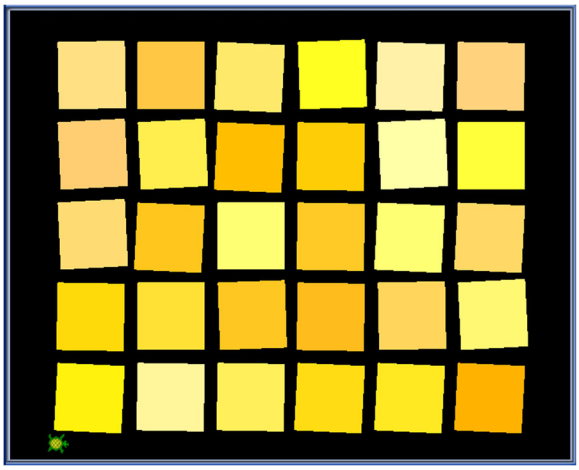

Fig. 11 Patio. Squares are placed at specific coordinates on the screen. There is some randomness in the color and orientation of the squares

put in rooms of different sizes. It would not know where the walls are unless it bumps into one. But when the turtle lives on a computer screen, we can specify a position using X and Y coordinates. TurtleArt includes Cartesian Geometry as well as Turtle geometry.

In both Titanic Rain and Patio, the elements of the composition—droplets or squares—are created using Turtle geometry. This allows the element to be drawn at different locations using the same code. The placement of the elements on the canvas is done using Cartesian Geometry.

Generative art with Scratch

Scratch is a programming environment and online community that is geared to young people ages nine to 15. It grows out of the same Logo tradition as TurtleArt and includes a similar drawing capability. But it is well suited to and more frequently used for creating animated stories, games, and multimedia productions. Scratch uses a blocks programming grammar that is similar to TurtleArt.

Instead of the single turtle in TurtleArt, Scratch has many "sprites"—objects that can assume a variety of shapes, known as "costumes," and move around the stage. Sprites can also draw, just like the TurtleArt turtle.

The Scratch Project *Squares* (Fig. 12) is similar to *Patio* in TurtleArt. But instead of having the turtle draw the squares, each square is a sprite.

Each sprite's costume was created in the Paint Editor that is part of Scratch. The sprites each have their own code (Fig. 13), which positions them on the stage and then sets the color and rotation randomly within constraints.[7]

In addition to creating uncertainty through randomness, Scratch programs may be altered by inputs from the outside world via the keyboard, mouse or touchpad, the computer's microphone, or video camera.

An example is Jiggle, a project that generates an ever-changing display of moving rectangles.[8] Each one is a clone of the original sprite, with a new one created every 0.25 s. They are in layers, with the most recent in front. Randomness is used to determine which of three rectangle shapes each clone assumes, how big it is, which way it

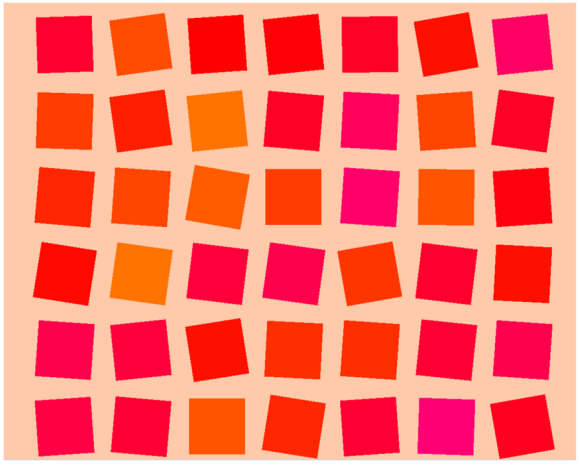

Fig. 12 Squares. The Scratch project Squares is visually similar to the TurtleArt project Patio, but is implemented differently

aims before starting to move, and how long it remains active before disappearing. But the color is determined not by randomness, but by the position of the mouse pointer. If the mouse is left untouched for a while, the pattern settles into a uniform color (Fig. 14).

When the mouse is moved, new rectangles appear in a different color (Fig. 15).

If the mouse is moved continuously, there will be rectangles of many colors (Fig. 16).

Motion Sensitive Squares[9] is a remix of the Squares project described above. What the video camera sees is displayed as a layer behind the array of sprites. When a sprite detects motion in the video image, it points in the direction of that motion.

The display generated by Drift[10] is also affected by motion that the video camera detects (Fig. 17).

The four rectangles are still when there is no motion in front of the camera, but each one is set in motion for a while when it detects a movement at its position. In this project, the video layer is set to a transparency of 100%, meaning that it is not visible, but motion is detected nonetheless. For a viewer, it is not immediately clear what is causing the rectangles to move.

With the use of additional hardware and using modified versions of Scratch one is not limited to works of art that are displayed on the computer screen. One may build structures and control them with Scratch programs. A full discussion of these

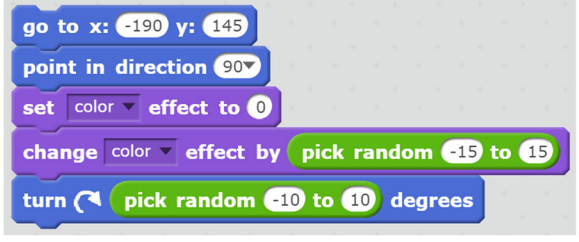

Fig. 13 Scratch Code from the Squares project. The same code is in each of the 42 sprites that make up the array of squares. The first block positions the sprite on the screen—the specific *X* and *Y* values are unique to each sprite. Constrained randomness is introduced into the color and orientation of each square

Fig. 14 The Scratch project Jiggle. After letting the program run for a while without moving the mouse, the ruslting image is all one color

possibilities is beyond the scope of this article, but we will mention one example. Light-Play (https://tinkering.exploratorium.edu/light-play) is an environment for exploring light, shadow, and motion. Scratch programs control light intensity and color while motors rotate objects to create moving patterns of light and shadow.

Comparing TurtleArt and Scratch

We have seen how both TurtleArt and Scratch can be used to explore generative art. In both environments the artist builds a system—a computer program—that generates the work of art. How are these two environments different?

TurtleArt is narrowly focused on creating drawings—two-dimensional static images that may be viewed on the screen or printed. The domain for Scratch is much broader, including animated displays.

TurtleArt is purely algorithmic. Everything that appears on the screen is generated by code. Scratch includes both algorithmic and non-algorithmic components. Graphic elements may be drawn using the built-in paint editor, or imported.

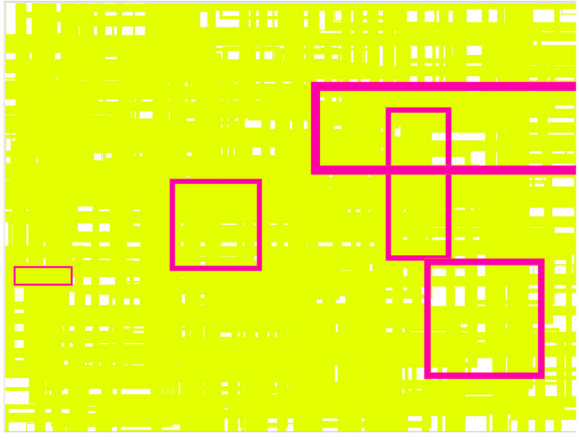

Fig. 15 Moving the mouse while Jiggle is running. The program detects the motion of the mouse and sets a new color based on the *X* coordinate of the mouse pointer's position on the screen

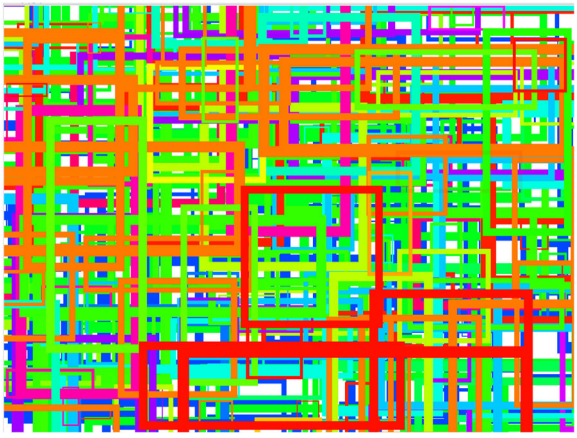

Fig. 16 Continuous mouse movement while Jiggle is running. The new rectangle colors changes with each movement of the mouse producing a multi-colored pattern

Scratch is a more complex environment than TurtleArt; so many more kinds of projects are possible.

TurtleArt and Scratch in the context of Logo development

Both TurtleArt and Scratch are inspired by, or descended from the decade-long development of the Logo programming language. We have seen how TurtleArt is a programming environment designed for artistic expression, specifically to create drawings, and for exploring generative art. It is also a vehicle for exploring and learning mathematics and programming. A guiding principle in Logo development has been to create programming environments that have a "low threshold and high ceiling." A beginner should be able to enter easily without obstacles or a big step up. But the language should also allow for sophisticated programming.

With the development of Scratch beginning in the mid-2000s, an additional design criterion was emphasized, that of "wide walls." People with differing interests should all be able to express themselves by creating projects in various domains—art, music, mathematics, science, storytelling, games, and animations (Resnick and Silverman 2005).

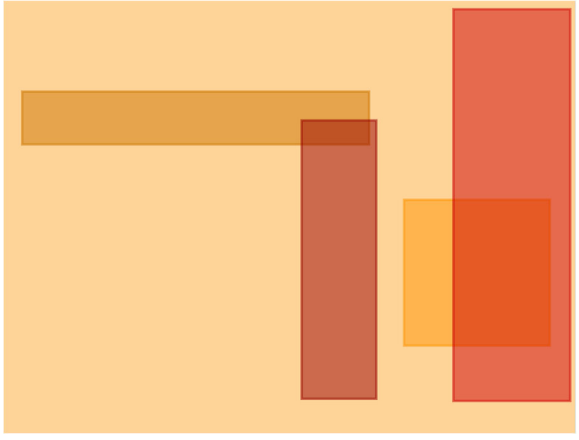

Fig. 17 The Scratch project Drift. The rectangles are set in motion when the computer's video camera detects motion

How does TurtleArt fit into this framework? It has a very low threshold, arguably lower than that of Scratch because of its simplicity. But as a programming language it is limited, so the ceiling is also low. And the walls are extremely narrow, focusing on a specific area of artistic expression. Brian Silverman, one of the developers of TurtleArt argues that this narrowness encourages "going deep" into the domain of algorithmic drawing. In part this is a result of the highly focused environment and lack of distractions.[11] But there are also details in the design of TurtleArt that make it especially well suited to the domain.[12]

We can also compare TurtleArt to the many versions of Logo that have been created over the past 50 years, most of which include turtle geometry and can be used in ways similar to TurtleArt. But most of these applications are more complex than TurtleArt, offering a greater range of possibilities for projects and explorations, but with a concurrent lack of focus.

One can look at the low threshold and high ceiling goal in different ways. An environment can be designed to be broadly inclusive, like Scratch. But one can also imagine a family of programming languages, with different dialects and vocabularies, but enough similarity between them so as to make it easy for people to move from one to the other.

One would choose an appropriate environment for the exploration or project at hand. For TurtleArt, that domain is generative art, specifically drawing, at the intersection of art, mathematics, and computing.

For Scratch, the domain is much broader encompassing many areas with an emphasis on animated stories, games, music, and multimedia projects. Within this range of possibilities, generative art explorations and projects involving mathematics and computing are possible. Since both TurtleArt and Scratch are members of the same broader family of computer programming environments, users may move comfortably between the two.

Endnotes

[1]Art that includes mathematical specification may or may not be algorithmic. For example, in classical Greek architecture, the proportions of the components of columns are precisely spelled out for each order, but this does not provide a step by step procedure for building the column. Weaving is an example of algorithmic art that has existed for thousands of years and has been automated for more than two centuries, initially with the Jacquard loom and its predecessors.

[2]TurtleArt is a product of the Playful Invention Company. The software, galleries of images, and tutorials are available from www.turtleart.org.

[3]Scratch https://scratch.mit.edu a project of the Lifelong Kindergarten Group at the MIT Media Lab.

[4]These workshops include *TurtleArt: the art of programming, the programming of art* (www.logofoundation.org/turtleart) and *Generating Surprise with Scratch and Turtle Art!* (www.logofoundation.org/genart). TurtleArt and Scratch are also used extensively in the Logo Summer Institutes, www.logofoundation.org/summer. Scratch projects from one of these workshops are in the Scratch Studio https://scratch.mit.edu/studios/2941611/.

[5]This example and the subsequent one, Patio, were created by Brian Silverman. These drawings and many more may be found in the galleries on the TurtleArt website—www.turtleart.org.

[6]The XY coordinates [0 0] are at the center of the canvas. *X* values are in the range of −350 to 350. *Y* values range from −260 to 260.

[7]The complete code for Patio may be seen at https://turtleart.org/programming/book1/imagepage.html?13. Click the yellow block icon at the right below the image. Squares may be seen on the Scratch website at https://scratch.mit.edu/projects/116485918/. Click "see inside" to view the code.

[8]Jiggle may be seen at https://scratch.mit.edu/projects/117332449/.

[9]Motion Sensitive Squares may be seen at https://scratch.mit.edu/projects/117776207.

[10]Drift may be seen at https://scratch.mit.edu/projects/117200862/.

[11]Conversations with the author during October 2016.

[12]For example, the floor tile pattern that appears earlier in this article can be drawn with TurtleArt as a pattern of repeated hexagons. As with any infinite tile pattern, when a border is placed around a portion of the design, some of the elements may appear only partially. TurtleArt allows one to draw a hexagon near the edge of the canvass with the turtle leaving the screen and returning as if the non-visible part of the shape is drawn beyond the border. The hexagon pattern could also be drawn in Scratch, but the turtle cannot leave the stage, so the tile pattern is disturbed at the edges. This is not a defect in Scratch, but rather a reflection of design criteria that are better suited to animation and game projects.

Acknowledgements
None

Funding
Does not apply

Availability of data and materials
Does not apply

Author's contributions
Michael Tempel is the sole author and is responsible for the 100% of the content.

Author's information
Michael Tempel is the president of the Logo Foundation, a nonprofit educational organization devoted to supporting educators, parents, and students in their engagement with creative computing.

Competing interests
The author declares that there are no competing interests.

Consent for publication
Does not apply

Ethics approval and consent to participate
Does not apply

Publisher's Note

Received: 10 September 2016 Accepted: 14 April 2017
Published online: 13 June 2017

References

Boytchev, Pavel, Logo Tree Project (2016), http://www.elica.net/download/papers/LogoTreeProject.pdf. Accesses 23 Mar 2017.

Galanter, P. (2003). *What is generative art?* http://www.philipgalanter.com/downloads/ga2003_paper.pdf. Accessed 23 Mar 2017.

What is Logo? (2014). http://el.media.mit.edu/logo-foundation/what_is_logo/index.html. Accesses 23 Mar 2017.

McCormack, J., Bown, O., Dorin, A., McCabe, J., Monro, G., & Whitelaw, M. (2014). Ten questions concerning generative computer art. *Leonardo, 47*(2), 135–141.
Papert, Seymour. Mindstorms (1981). New York, NY: Basic Books, Chapter 3.
Resnick, M., & Silverman, B. (2005). *Some reflections on designing construction kits for kids*. http://web.media.mit.edu/~mres/papers/IDC-2005.pdf. Accessed 23 Mar 2017.
Tempel, M. (2013). Blocks programming. *CSTA Voice, 9*(1), 3–4.